CPSC 532D — 16. NEURAL TANGENT KERNELS

Danica J. Sutherland

University of British Columbia, Vancouver

Fall 2023

Continuing our study of nonconvex optimization, we're going to see one setting today where we can prove global optimization for certain deep networks.

Initially, there were a series of papers that proved this in some special cases; later, it was realized that they all shared a common structure, which amounts to the following: if you initialize a very wide network appropriately and minimize the square loss with a small learning rate, your model becomes equivalent to kernel regression, with a particular kernel defined by the architecture.

1 NEURAL TANGENT KERNELS

To motivate things, let's begin with a very simple network:

$$h(x; W) = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} a_j \sigma(w_j \cdot x),$$

We could of course absorb the $\sqrt{N}$ into the a_j. Writing it like this scales the gradients w.r.t. a_j differently, and is called the NTK parameterization; *it makes some things simpler.*

where the a_j are *fixed* and the $w_j \in \mathbb{R}^N$ are the rows of a weight matrix $W \in \mathbb{R}^{N \times d}$, viewed as column vectors. We're writing W as a parameter to the function because want to emphasize that we're changing W, not x.

Now, let's see what happens if we take a first-order Taylor expansion (a linearization) of *h with respect to* W, around some point $\tilde{W}$. (We're going to pick $\tilde{W}$ to be our starting point for gradient descent.) This function will be linear in W but nonlinear in x; it should be a reasonable approximation if $W \approx \tilde{W}$.

$$\begin{aligned} h_{\tilde{W}}(x; W) &= h(x; \tilde{W}) + \langle \nabla_W h(x; \tilde{W}), W - \tilde{W} \rangle \qquad (1) \\ &= \frac{1}{\sqrt{N}} \sum_{j=1}^{N} a_j \left[\sigma(\tilde{w}_j \cdot x) + \langle \sigma'(\tilde{w}_j \cdot x) x, w_j - \tilde{w}_j \rangle \right] \\ &= \frac{1}{\sqrt{N}} \sum_{j=1}^{N} a_j \Big[\underbrace{\sigma(\tilde{w}_j \cdot x) - \sigma'(\tilde{w}_j \cdot x) \tilde{w}_j \cdot x}_{\text{constant in W}} + \underbrace{\sigma'(\tilde{w}_j \cdot x) x \cdot w_j}_{\text{linear in W}} \Big]. \end{aligned}$$

We can either think of this as a matrix with the Frobenius inner product $\langle A, B \rangle = \sum_{ij} A_{ij} B_{ij}$, or of flattening the parameters into a vector and using regular gradients.

Notice that we have $\mathrm{ReLU}(t) = \mathrm{ReLU}'(t) t$ for all t (even zero), and so for ReLU activations the term that is constant in W vanishes. In that case, we see that

$$h_{\tilde{W}}(x; W) = \langle \nabla_W h(x; \tilde{W}), W \rangle.$$

But, if we write $\phi_{\tilde{W}}(x) = \nabla_W h(x; \tilde{W})$, the function $h_{\tilde{W}}$ is of the form $\langle W, \phi_{\tilde{W}}(x) \rangle$, and in particular the set of achievable functions

$$\{x \mapsto h_{\tilde{W}}(x; W) : W \in \mathbb{R}^{N \times d}\}$$

For more, visit https://cs.ubc.ca/~dsuth/532D/23w1/.

is exactly the RKHS of the kernel

$$k_{\tilde{W}}(x, x') = \langle \phi_{\tilde{W}}(x), \phi_{\tilde{W}}(x') \rangle = \langle \nabla_W h(x; \tilde{W}), \nabla_W h(x'; \tilde{W}) \rangle. \tag{2}$$

Even if we don't have ReLU activations, recalling (1) we can see that the *residual* $h_{\tilde{W}}(x; W) - h(x; \tilde{W})$ is an element of that same RKHS.

But, so what? This linearization is only going to be accurate when $W \approx \tilde{W}$, it doesn't tell us anything about actually training a network. ...unless?

1.1 *Approximation quality*

PROPOSITION 1. *For the h and $h_{\tilde{W}}$ defined above, assume σ is β-smooth. Then*

$$\left| h(x; W) - h_{\tilde{W}}(x; W) \right| \le \frac{\beta}{2\sqrt{N}} (\max_j |a_j|) \|x\|^2 \left\| W - \tilde{W} \right\|_F^2 .$$

Proof. Recall (Proposition 3 of the nonconvex optimization notes) that β-smooth functions deviate from their linearizations by at most a quadratic. Thus

$$\begin{aligned} \left| h(x; W) - h_{\tilde{W}}(x; W) \right| &\le \frac{1}{\sqrt{N}} \sum_{j=1}^N |a_j| \left| \sigma(w_j \cdot x) - \sigma(\tilde{w}_j \cdot x) - \sigma'(\tilde{w}_j \cdot x)(w_j \cdot x - \tilde{w}_j \cdot x) \right| \\ &\le \frac{1}{\sqrt{N}} \sum_{j=1}^N |a_j| \frac{1}{2} \beta (w_j \cdot x - \tilde{w}_j \cdot x)^2 \\ &\le \frac{\beta}{2\sqrt{N}} (\max_j |a_j|) \sum_{j=1}^N \left\| w_j - \tilde{w}_j \right\|^2 \|x\|^2 . \end{aligned}$$

□

So, if N is large enough, the linearization is good even for a large radius of parameters around $\tilde{W}$, no matter what the data looks like (as long as it's bounded) and no matter what $\tilde{W}$ looks like.

One thing to keep in mind, though, is that $\left\| W - \tilde{W} \right\|_F$ is also changing meaning with N: in fact, $\mathbb{E} \left\| \tilde{W} \right\|_F^2 = \mathbb{E} \sum_{j=1}^N \sum_{k=1}^d \tilde{W}_{jk}^2 = Nd$, so with standard Gaussian initializations the linearization potentially even gets worse at $W = 0$. It's thus not really obvious yet that this approximation is a "good idea." It'll turn out, though, that in the high-width regime, the optimization process doesn't have to move very much, and everything works out.

The result being *so* general in $\tilde{W}$ is special for this very simple network; even for the same architecture but with ReLU activations, the analysis is a lot nastier.

PROPOSITION 2 (Lemma 4.1 of [Tel]). *For any radius $B \ge 0$, for any fixed $x \in \mathbb{R}^d$ with $\|x\| \le 1$, suppose we pick $\tilde{W}$ with entries i.i.d. standard normal. Then, with probability at least $1 - \delta$ over the draw of $\tilde{W}$, it holds that for all $W \in \mathbb{R}^{N \times d}$ with $\left\| W - \tilde{W} \right\|_F \le B$ that*

$$\left| h(x; W) - h_{\tilde{W}}(x; W) \right| \le \frac{2B^{4/3} + B \log(1/\delta)^{1/4}}{N^{1/6}}.$$

For deeper networks, the result is even more complicated and degrades with depth, but it's still going to be true that with appropriate Gaussian initializations the linearization works for an increasing radius.

1.2 *The neural tangent kernel*

What does the kernel (2) look like? For this shallow network, it's

$$\begin{aligned}
k_{\tilde{W}}(x,x') &= \langle \phi_{\tilde{W}}(x), \phi_{\tilde{W}}(x') \rangle = \langle \nabla_W h(x;\tilde{W}), \nabla_W h(x';\tilde{W}) \rangle \\
&= \left\langle \begin{bmatrix} a_1 x^\top \sigma'(\tilde{w}_1 \cdot x)/\sqrt{N} \\ \vdots \\ a_N x^\top \sigma'(\tilde{w}_N \cdot x)/\sqrt{N} \end{bmatrix}, \begin{bmatrix} a_1 (x')^\top \sigma'(\tilde{w}_1 \cdot x')/\sqrt{N} \\ \vdots \\ a_N (x')^\top \sigma'(\tilde{w}_N \cdot x')/\sqrt{N} \end{bmatrix} \right\rangle \\
&= \frac{1}{N} \sum_{j=1}^N a_j^2 \langle x\sigma'(\tilde{w}_j \cdot x), x'\sigma'(\tilde{w}_j \cdot x') \rangle \\
&= x \cdot x' \left[\frac{1}{N} \sum_{j=1}^N a_j^2 \sigma'(\tilde{w}_j \cdot x)\sigma'(\tilde{w}_j \cdot x') \right].
\end{aligned}$$

The $1/\sqrt{N}$ scaling hopefully makes sense now: if we also use the common setup $a_j \in \{\pm 1\}$ so that $a_j^2 = 1$, as $N \to \infty$, this converges almost surely to the kernel

$$k_\infty(x,x') = x \cdot x' \mathop{\mathbb{E}}_{\tilde{w}\sim\mathcal{N}(0,I)} [\sigma'(\tilde{w}\cdot x)\sigma'(\tilde{w}\cdot x')].$$

We'll call $k_{\tilde{W}}$ the *empirical neural tangent kernel at* $\tilde{W}$ and k_∞ the *infinite neural tangent kernel*. Terminology here isn't quite standardized, though.

For the ReLU, $\sigma'(t) = \mathbb{1}(t>0)$, and so k_∞ is $x \cdot x'$ times

$$\begin{aligned}
\mathop{\mathbb{E}}_{\tilde{w}\sim\mathcal{N}(0,I)} [\sigma'(\tilde{w}\cdot x)\sigma'(\tilde{w}\cdot x')] &= \mathop{\Pr}_{\tilde{w}\sim\mathcal{N}(0,I)} (\tilde{w}\cdot x > 0 \text{ and } \tilde{w}\cdot x' > 0) \\
&= \mathop{\Pr}_{v\sim\text{Unif(unit sphere)}} \left(v \cdot \frac{x}{\|x\|} > 0 \text{ and } v \cdot \frac{x'}{\|x'\|} > 0 \right) \\
&= \frac{\pi - \arccos\left(\frac{x\cdot x'}{\|x\|\|x'\|}\right)}{2\pi}.
\end{aligned}$$

Draw a picture for this...

In fact, a slight tweak to this kernel is even universal [Tel, Theorem 4.1].

1.3 *More general networks*

For more general architectures, we still use the linearization

$$h_{\tilde{w}}(x;w) = h(x;\tilde{w}) + \langle \nabla_w h(x;\tilde{w}), w - \tilde{w} \rangle,$$

where now w represents *all* the parameters in the network collected into a vector. This gives us the same kind of empirical neural tangent kernel

$$k_{\tilde{w}}(x,x') = \langle \nabla_w h(x;\tilde{w}), \nabla_w h(x';\tilde{w}) \rangle. \tag{3}$$

Even for complex deep architectures (including convolutions, pooling, even attention), if we scale things according to *fan-out* "NTK parameterizations," $k_{\tilde{w}}$ still converges to some k_∞ almost surely as it becomes infinitely wide [Yan19]. It's also often possible (though computationally expensive) to exactly compute this kernel; the standard software is the `neural-tangents` library [Nov+20].

2 OPTIMIZATION

So, we know that the set of "nearby" updates to the initialization $\tilde{w}$ is described by the set of functions in an RKHS. But maybe we break out of the "nearby" set immediately, and this is all pointless.

In some regimes, it's true that we break the NTK approximation immediately, in practice; e.g. we can check numerically whether k_{w_0} and $k_{w_{100}}$ are very different. But, if we set things up in the right way, this doesn't happen, and the whole gradient descent process stays near enough to w_0 that the RKHS is a good approximation.

To see this, we'll consider square loss and take the limit of gradient descent as the step size η goes to zero, which is known as *gradient flow*. A lot of the time, the continuous limit is much easier to handle, and then you check that the discretization into actual gradient descent doesn't take you too far away from the continuous version.

In this case, we have

$$\frac{\mathrm{d}w_t}{\mathrm{d}t} = -\eta \nabla_w \mathrm{L}_\mathrm{S}(h(\cdot; w_t)) = -\frac{2\eta}{m}\sum_{i=1}^{m}(h(x_i; w_t) - y_i)\nabla_w h(x_i; w_t), \tag{4}$$

and so if we track the training set predictions in particular we get

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}h(x_j; w_t) &= \left\langle \nabla h(x_j; w_t), \frac{\mathrm{d}w_t}{\mathrm{d}t}\right\rangle \\ &= -\frac{2\eta}{m}\sum_{i=1}^{m}(f(x_i; w_t) - y_i)\langle \nabla_w h(x_i; w_t), \nabla w h(x_j; w_t)\rangle \\ &= -\frac{2\eta}{m}\sum_{i=1}^{m}(f(x_i; w_t) - y_i)k_{w_t}(x_i, x_j).\end{aligned}$$

Summarizing all of these in vector form, the vector of predictions on the training set $h|_{\mathrm{S}_x}(t) = (h(x_1; w_t), \ldots, h(x_m; w_t))$ evolves as

$$\frac{\mathrm{d}}{\mathrm{d}t}h|_{\mathrm{S}_x}(t) = -\frac{2\eta}{m}k_{w_t}|_{\mathrm{S}_x}(h|_{\mathrm{S}_x}(t) - \mathrm{S}_y), \tag{5}$$

where $k_{w_t}|_{\mathrm{S}_x}$ is the $m \times m$ kernel matrix $[k(x_i, x_j)]_{ij}$, and $\mathrm{S}_y = (y_1, \ldots, y_m) \in \mathbb{R}^m$.

Suppose that k_{w_t} is constant in t. Then this would be *exactly the same dynamics* as "kernel gradient descent" for kernel regression, which we'll define now.

To use kernel gradient descent, it'll be convenient to define the *outer product* for Hilbert spaces, also called a tensor product. For $a \in \mathcal{F}_1$, $b \in \mathcal{F}_2$, we can define $a \otimes b : \mathcal{F}_2 \to \mathcal{F}_1$ as a linear operator by $[a \otimes b]b' = \langle b, b'\rangle a$. For $\mathbb{R}^d$, this is just $(ab^\mathsf{T})b' = a(b^\mathsf{T}b')$.

Kernel gradient descent for square loss then gives that

Note that this is not *the same as using the representer theorem and doing gradient descent on the coefficients α! That's a different parameterization, giving different gradients.*

$$\begin{aligned}
L_S(h) &= \frac{1}{m}\sum_{i=1}^{m}(h(x_i)-y_i)^2 \\
&= \frac{1}{m}\sum_{i=1}^{m}\Big(\langle h, \varphi(x_i)\rangle\langle\varphi(x_i), h\rangle - 2y_i\langle\varphi(x_i), h\rangle + y_i^2\Big) \\
&= \left\langle h, \left[\frac{1}{m}\sum_{i=1}^{m}\varphi(x_i)\otimes\varphi(x_i)\right] h\right\rangle - 2y_i\left\langle\frac{1}{m}\sum_{i=1}^{m}\varphi(x_i), h\right\rangle + \frac{1}{m}\sum_{i=1}^{m}y_i^2 \\
\frac{\mathrm{d}h}{\mathrm{d}t} &= -\eta\nabla_h L_S(h) = -\left[\frac{2\eta}{m}\sum_{i=1}^{m}\varphi(x_i)\otimes\varphi(x_i)\right]h + \frac{2\eta}{m}\sum_{i=1}^{m}y_i\varphi(x_i) \\
&= -\frac{2\eta}{m}\sum_{i=1}^{m}h(x_i)\varphi(x_i) + \frac{2\eta}{m}\sum_{i=1}^{m}y_i\varphi(x_i) \\
&= -\frac{2\eta}{m}\sum_{i=1}^{m}(h(x_i)-y_i)\varphi(x_i) \qquad (6) \\
\frac{\mathrm{d}}{\mathrm{d}t}h(x_j) &= \left\langle\frac{\mathrm{d}h}{\mathrm{d}t}, \varphi(x_j)\right\rangle = -\frac{2\eta}{m}\sum_{i=1}^{m}(h(x_i)-y_i)k(x_i,x_j),
\end{aligned}$$

and summarizing in matrix form this becomes

$$\frac{\mathrm{d}}{\mathrm{d}t}h|_{S_x}(t) = -\frac{2\eta}{m}k|_{S_x}(h|_{S_x} - S_y). \qquad (7)$$

Indeed, (5) and (7) agree if $k_{w_t}|_{S_x}$ is constant throughout training; so do (4) and (6), recalling the definition of the kernel (3). In fact, though, we can solve this ordinary differential equation (7) analytically [JGH18]; we get for an arbitrary prediction point that

$$h_t(x) = h_0(x) + [k(x,x_i)]_{i=1}^{m}(k|_{S_x})^{-1}\left(I - e^{-\eta t\, k|_{S_x}}\right)(S_y - h_0|_{S_x}),$$

where the matrix exponential $e^A = I + A + \frac{1}{2}A^2 + \dots$ can be found by exponentiating the singular values of A. In the limit as $t\to\infty$, $e^{-\eta t k|_{S_x}}\to 0$ as long as $k|_{S_x}$ is nonsingular (which it is for universal kernels). We can also see then that $h_t|_{S_x}\to S_y$.

This becomes equivalent to $\lim_{\lambda\to 0}\arg\min_{h\in\mathcal{F}} L_S(h) + \lambda\|h-h_0\|_{\mathcal{F}}^2$, sometimes called kernel "ridgeless" regression; it's also like GP regression with a prior mean of h_0.

In the limit as the network becomes infinitely wide, for appropriate initializations, $k_{w_t}|_{S_x}$ does in fact stay constant through training [Lee+19; Aro+19; COB19; YL21]. For example, Theorem 3.2 of Arora et al. [Aro+19] shows a closeness guarantee with finite width. The proof basically amounts to showing that parameters don't change much, so the empirical NTK doesn't change much, and so gradients throughout the network training are close to gradients from the kernel process.

This behaviour is very much a function of having the right scaling, though. For "fan-in" parameterizations that people usually actually use in practice, the kernel blows up, but it does so in a way where regression still makes sense [Lee+19; MBS23]. If you choose other scalings, it doesn't happen at all; Chizat, Oyallon, and Bach [COB19] cover this point of view, arguing that the parameter scaling amounts to "zooming in" on the Taylor expansion so the linearization works well. Yang and Hu [YH22] also study scaling regimes other than just the kernel one.

Dealing with the random initialization $h_0(x)$ can be a problem, sometimes; it takes a while for gradient descent to counteract the noise there. Sometimes people scale the network output by a small $\varepsilon > 0$ (which also scales the gradient, but that's good too), or use special symmetric initializations to their network parameters which guarantee $h_0(x) = 0$.

3 DOES THIS MEAN DEEP LEARNING IS SOLVED?

So, in these regimes, you can avoid training a network at all and just use a special kernel. That kernel is indeed often a good kernel for particular problems [Aro+20].

But, there are problems that deep learning can solve provably faster than *any* kernel method. Essentially, the NTK regime doesn't allow for feature learning, since the kernel is fixed.

One concrete example is learning a single ReLU: Yehudai and Shamir [YS19] show that kernel methods cannot learn a single ReLU unit $\operatorname{ReLU}(w \cdot x + b)$ in square loss without RKHS norm exponential in the dimension, which implies (roughly) exponential sample complexity – but gradient descent can get it with polynomially many samples.

Malach et al. [MKAS21] give a nice characterization on when these gaps are and aren't possible.

Overall, it's definitely *not* the case that the practical success of deep learning can be explained by infinite NTK behaviour. Even so, the infinite NTK is still useful to know about:

- As far as I know, they're the only proofs that gradient descent works in nonlinear deep nets.
- It's actually a reasonable approximation in certain regimes.
- Infinite NTKs are sometimes practically useful [Aro+20; JNWB21].

Also, I think the *empirical* NTK is even more useful: I've given a talk called "A Defense of (Empirical) Neural Tangent Kernels". Basically, the empirical NTK can be a really useful tool for understanding what's happening "locally" during training, even when it doesn't stay constant over the whole course of training. This can be theoretical [RGS22; RGBS23], for "empirical science" understanding [For+20; MBS23], or even purely practical [WHS22; MBS22].

REFERENCES

[Aro+19] Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. "On Exact Computation with an Infinitely Wide Neural Net." *NeurIPS*. 2019. arXiv: 1904.11955.

[Aro+20] Sanjeev Arora, Simon S. Du, Zhiyuan Li, Ruslan Salakhutdinov, Ruosong Wang, and Dingli Yu. "Harnessing the Power of Infinitely Wide Deep Nets on Small-data Tasks." *ICLR*. 2020. arXiv: 1910.01663.

[COB19] Lenaic Chizat, Edouard Oyallon, and Francis Bach. "On Lazy Training in Differentiable Programming." *NeurIPS*. 2019. arXiv: 1812.07956.

[For+20] Stanislav Fort, Gintare Karolina Dziugaite, Mansheej Paul, Sepideh Kharaghani, Daniel M. Roy, and Surya Ganguli. "Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the Neural Tangent Kernel." *NeurIPS*. 2020. arXiv: 2010.15110.

[JGH18] Arthur Jacot, Franck Gabriel, and Clément Hongler. "Neural Tangent Kernel: Convergence and Generalization in Neural Networks." *NeurIPS*. 2018. arXiv: 1806.07572.

[JNWB21] Sheng Jia, Ehsan Nezhadarya, Yuhuai Wu, and Jimmy Ba. "Efficient Statistical Tests: A Neural Tangent Kernel Approach." *ICML*. 2021.

[Lee+19] Jaehoon Lee, Lechao Xiao, Samuel S Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. "Wide neural networks of any depth evolve as linear models under gradient descent." *NeurIPS*. 2019. arXiv: 1902.06720.

[MBS22] Mohamad Amin Mohamadi, Wonho Bae, and Danica J. Sutherland. "Making Look-Ahead Active Learning Strategies Feasible with Neural Tangent Kernels." *NeurIPS*. 2022. arXiv: 2206.12569.

[MBS23] Mohamad Amin Mohamadi, Wonho Bae, and Danica J. Sutherland. "A Fast, Well-Founded Approximation to the Empirical Neural Tangent Kernel." *ICML*. 2023. arXiv: 2206.12543.

[MKAS21] Eran Malach, Pritish Kamath, Emmanuel Abbe, and Nathan Srebro. "Quantifying the Benefit of Using Differentiable Learning over Tangent Kernels." *ICML*. 2021. arXiv: 2103.01210.

[Nov+20] Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. "Neural Tangents: Fast and Easy Infinite Neural Networks in Python." *ICLR*. 2020. arXiv: 1912.02803.

[RGBS23] Yi Ren, Shangmin Guo, Wonho Bae, and Danica J. Sutherland. "How to prepare your task head for finetuning." *ICLR*. 2023. arXiv: 2302.05779.

[RGS22] Yi Ren, Shangmin Guo, and Danica J. Sutherland. "Better Supervisory Signals by Observing Learning Paths." *ICLR*. 2022. arXiv: 2203.02485.

[Tel] Matus Telgarsky. *Deep learning theory lecture notes*. Version: 2021-10-27 v0.0-e7150f2d (alpha). 2021.

[WHS22] Alexander Wei, Wei Hu, and Jacob Steinhardt. "More Than a Toy: Random Matrix Models Predict How Real-World Neural Representations Generalize." *ICML*. 2022. arXiv: 2203.06176 [cs.LG].

[Yan19] Greg Yang. "Tensor Programs I: Wide Feedforward or Recurrent Neural Networks of Any Architecture are Gaussian Processes." *NeurIPS*. 2019. arXiv: 1910.12478.

[YH22] Greg Yang and Edward J. Hu. "Feature Learning in Infinite-Width Neural Networks." *ICML*. 2022. arXiv: 2011.14522.

[YL21] Greg Yang and Etai Littwin. "Tensor Programs IIb: Architectural Universality of Neural Tangent Kernel Training Dynamics." *ICML*. 2021. arXiv: 2105.03703.

[YS19] Gilad Yehudai and Ohad Shamir. "On the Power and Limitations of Random Features for Understanding Neural Networks." *NeurIPS*. 2019. arXiv: 1904.00687.